On the Weakest Failure Detector for Quiescent Reliable Communication*

Marcos Kawazoe Aguilera Wei Chen Sam Toueg

Department of Computer Science
Upson Hall, Cornell University
Ithaca, NY 14853-7501, USA.
aguilera,weichen,sam@cs.cornell.edu

July 1997

Abstract

We consider the problem of achieving reliable communication with *quiescent* algorithms (i.e., algorithms that eventually stop sending messages) in asynchronous systems with process crashes and lossy links, and show that, among failure detectors with bounded output size, $\Diamond\mathcal{P}$ is the weakest one that can be used to solve this problem. Combined with a result in [ACT97], this shows that failure detectors that are commonly used in practice, i.e., those that output lists of suspects, are not always the best ones to solve a problem.

1 Introduction

In [ACT97], we considered the problem of achieving reliable communication with *quiescent* algorithms (i.e., algorithms that eventually stop sending messages) in asynchronous systems with process crashes and lossy links. In that paper, we show that this problem can be solved with *heartbeat*, a failure detector that can be implemented in such systems (without timeouts). Unlike failure detectors that are commonly considered in the literature and used in practice, heartbeat does not output lists of processes suspected to have failed. Instead, it outputs a vector of counters, one for each process, such that the counter of a process increases without bound if and only if that process does not fail.

So quiescent reliable communication can be achieved with heartbeat, a failure detector that is implementable but has unbounded output size. This raises the following question: Can we achieve quiescent reliable communication with a failure detector that is implementable *and* has bounded output size? In this paper we prove that the answer is *no*: among failure detectors with bounded output size, the weakest one that can be used to solve this problem is the *Eventually Perfect failure*

*Research partially supported by NSF grant CCR-9402896, by ARPA/ONR grant N00014-96-1-1014, and by an Olin Fellowship.

detector $\Diamond\mathcal{P}$ — a failure detector that cannot be implemented in asynchronous systems with failures (an implementation would violate a known impossibility result [FLP85, CT96]).

Thus, a failure detector with bounded output size is either (a) too weak to achieve quiescent reliable communication, or (b) not implementable. Combined with the results in [ACT97], this shows that failure detectors that are commonly used in practice, i.e., those that output lists of suspects, are not always the best ones to solve a problem: their power or applicability is limited.

In this paper, we prove our result with respect to a problem that we call *Single-Shot Reliable Send and Receive*. This is a weaker version of the reliable communication problems considered in [ACT97], and thus the result in this paper immediately apply to those problems as well. We assume that links can lose only a finite number of messages, and the network is completely connected. It is straightforward to verify that the result in this paper also holds for *fair* links [ACT97] and *fair lossy* links [BCBT96], and also for networks that are not completely connected as defined in [ACT97].

The paper is organized as follows. Our model is described in Section 2. In Section 3, we explain what it means for a failure detector to be weaker than another one. Section 4 defines the reliable communication problem that we consider. In Section 5, we prove our result under some reasonable simplifying assumption. In the Appendix we drop this assumption and give the full proof.

2 Model

We consider asynchronous message-passing distributed systems in which there are no timing assumptions. In particular, we make no assumptions on the time it takes to deliver a message, or on relative process speeds. The system consists of a set of n processes $\Pi = \{1, 2, \ldots, n\}$ that are completely connected by point-to-point (bidirectional) links. The system can experience both process failures and link failures. A process can fail by crashing, and a link can fail by dropping a finite number of messages. Our model, described in the rest of this section, is based on the one in [CHT96].

We assume the existence of a discrete global clock — this is merely a fictional device to simplify the presentation and processes do not have access to it. We take the range $\mathcal{T}$ of the clock's ticks to be the set of natural numbers.

2.1 Failure Patterns

Processes can fail by crashing, i.e., by halting prematurely. A *failure pattern* F is a function from $\mathcal{T}$ to 2^{Π}. Intuitively, $F(t)$ denotes the set of processes that have crashed through time t. Once a process crashes, it does not "recover", i.e., $\forall t : F(t) \subseteq F(t+1)$. We define $\text{crashed}(F) = \bigcup_{t \in \mathcal{T}} F(t)$ and $\text{correct}(F) = \Pi \setminus \text{crashed}(F)$. If $p \in \text{crashed}(F)$ we say p *crashes (or is faulty) in* F and if $p \in \text{correct}(F)$ we say p is *correct in* F.

Links can fail by dropping a finite number of messages. This is made more precise in Section 2.3.

2.2 Failure Detectors and $\Diamond\mathcal{P}$

Each process has access to a local failure detector module that provides (possibly incorrect) information about the failure pattern that occurs in an execution. A *failure detector history H with range $\mathcal{R}$* is a function from $\Pi \times \mathcal{T}$ to $\mathcal{R}$. $H(p,t)$ is the output value of the failure detector module of process p at time t. *A failure detector $\mathcal{D}$* is a function that maps each failure pattern F to a *set* of failure detector histories with range $\mathcal{R}_\mathcal{D}$ (where $\mathcal{R}_\mathcal{D}$ denotes the range of the failure detector output of $\mathcal{D}$). $\mathcal{D}(F)$ denotes the set of possible failure detector histories permitted by $\mathcal{D}$ for the failure pattern F. Clearly, if the output of $\mathcal{D}$ has bounded size (it can be represented with a bounded number of bits) then the range of $\mathcal{D}$ is finite.

We now define the *eventually perfect failure detector* $\Diamond\mathcal{P}$ [CT96].[1] Each failure detector module of $\Diamond\mathcal{P}$ outputs a *set of processes* that are suspected to have crashed, i.e., $\mathcal{R}_{\Diamond\mathcal{P}} = 2^\Pi$. For each failure pattern F, $\Diamond\mathcal{P}(F)$ is the set of all failure detector histories H with range $\mathcal{R}_{\Diamond\mathcal{P}}$ that satisfy the following properties:

- *Strong Completeness*: Eventually every process that crashes is permanently suspected by every correct process. More precisely:

$$\exists t \in \mathcal{T}, \forall p \in \text{crashed}(F), \forall q \in \text{correct}(F), \forall t' \geq t : p \in H(q,t')$$

- *Eventual Strong Accuracy*: There is a time after which correct processes are not suspected by any correct process. More precisely:

$$\exists t \in \mathcal{T}, \forall t' \geq t, \forall p,q \in \text{correct}(F) : p \notin H(q,t')$$

2.3 Runs of Algorithms

An algorithm A is a collection of n (possibly infinite-state) deterministic automata, one for each process in the system. Computation proceeds in atomic *steps* of A. In each step, a process may: receive a message from a process, get an external input, query its failure detector module, undergo a state transition, send a message to a neighbor, and issue an external output.

A *run of algorithm A using failure detector $\mathcal{D}$* is a tuple $R = (F, H_\mathcal{D}, I, S, T)$ where F is a failure pattern, $H_\mathcal{D} \in \mathcal{D}(F)$ is a history of failure detector $\mathcal{D}$ for failure pattern F, I is an initial configuration of A, S is an infinite sequence of steps of A, and T is an infinite list of increasing time values indicating when each step in S occurs.

A run must satisfy some properties for every process p: If p has crashed by time t, i.e., $p \in F(t)$, then p does not take a step at any time $t' \geq t$; if p is correct, i.e., $p \in \text{correct}(F)$, then p takes an infinite number of steps; and if p takes a step at time t and queries its failure detector, then p gets $H_\mathcal{D}(p,t)$ as a response.

A run must also satisfy the following "link properties" for every pair of processes p and q:

- *Uniform Integrity*: For all $k \geq 1$, if q receives a message m from p exactly k times by time t, then p sent m to q at least k times before time t;

[1] In [CT96], $\Diamond\mathcal{P}$ denotes a *class* of failure detectors.

- *Finite Loss*: If q is correct, the number of messages sent by p to q that are not received by q is finite.

These properties model the behavior of the "link" from p to q. Intuitively, Uniform Integrity ensures that this link does not create or duplicate messages. Finite Loss ensures that there is a time after which every message sent through this link is eventually received (provided q is correct).

2.4 Environments and Problem Solving

The correctness of an algorithm may depend on certain assumptions on the "environment", e.g., the maximum number of processes that may crash. For example, a consensus algorithm may need the assumption that a majority of processes is correct. Formally, an *environment* $\mathcal{E}$ is a set of failure patterns.

A *problem* P is defined by properties that sets of runs must satisfy. An *algorithm* A *solves problem* P *using a failure detector* $\mathcal{D}$ *in environment* $\mathcal{E}$ if the set of all runs $R = (F, H_{\mathcal{D}}, I, S, T)$ of A using $\mathcal{D}$ where $F \in \mathcal{E}$ satisfies the properties required by P.

3 Failure Detector Transformations

As explained in [CT96, CHT96], failure detectors can be compared via algorithmic transformations. We now explain what it means for an algorithm $T_{\mathcal{D}\to\mathcal{D}'}$ to transform a failure detector $\mathcal{D}$ into another failure detector $\mathcal{D}'$ in an environment $\mathcal{E}$. Algorithm $T_{\mathcal{D}\to\mathcal{D}'}$ uses $\mathcal{D}$ to maintain a variable $\mathcal{D}'_p$ at every process p. This variable, reflected in the local state of p, emulates the output of $\mathcal{D}'$ at p. Let $H_{\mathcal{D}'}$ be the history of all the $\mathcal{D}'$ variables in a run R of $T_{\mathcal{D}\to\mathcal{D}'}$, i.e., $H_{\mathcal{D}'}(p,t)$ is the value of $\mathcal{D}'_p$ at time t in run R. Algorithm $T_{\mathcal{D}\to\mathcal{D}'}$ *transforms* $\mathcal{D}$ *into* $\mathcal{D}'$ *in* $\mathcal{E}$ if and only if for every $F \in \mathcal{E}$ and every run $R = (F, H_{\mathcal{D}}, I, S, T)$ of $T_{\mathcal{D}\to\mathcal{D}'}$ using $\mathcal{D}$, we have $H_{\mathcal{D}'} \in \mathcal{D}'(F)$. Intuitively, since $T_{\mathcal{D}\to\mathcal{D}'}$ is able to use $\mathcal{D}$ to emulate $\mathcal{D}'$, $\mathcal{D}$ provides at least as much information about process failures as $\mathcal{D}'$ does, and we say that $\mathcal{D}'$ *is weaker than* $\mathcal{D}$ *in* $\mathcal{E}$.

Note that, in general, $T_{\mathcal{D}\to\mathcal{D}'}$ need not emulate *all* the failure detector histories of $\mathcal{D}'$ (in environment $\mathcal{E}$); what we do require is that all the failure detector histories it emulates be histories of $\mathcal{D}'$ (in that environment).

4 Single-Shot Reliable Send and Receive

[ACT97] considers four reliable communication problems: two point-to-point ones, called *[quasi] reliable send and receive*, and two broadcast ones, called *[uniform] reliable broadcast*. We prove our result with respect to a simpler problem, called *Single-Shot Reliable Send and Receive*. Since this problem is weaker than those in [ACT97], our result also applies to these four problems.

The Single-Shot Reliable Send and Receive problem is defined in terms of two communication

primitives, called *Send* and *Receive*[2]. Each process can Send a single bit once to one process of its choice, if it wishes to do so (but it is also possible that no process in the system ever Sends any bit). The Send and Receive primitives must satisfy the following property. For any two correct processes p and q, and any $b \in \{0,1\}$: p Sends b to q if and only if q Receives b from p.

An implementation $\mathcal{I}$ of Send and Receive is quiescent if it sends only a finite number of messages throughout the network.

5 Main Result

Our goal is to show that $\Diamond\mathcal{P}$ is the weakest failure detector with finite range that can be used to achieve quiescent reliable communication. To do so, we focus on the Single-Shot Reliable Send and Receive problem, and prove that if a failure detector $\mathcal{D}$ with finite range can be used to solve this problem with a quiescent algorithm, then $\mathcal{D}$ can be transformed to $\Diamond\mathcal{P}$. Since this problem is weaker than the four problems defined in [ACT97], we conclude that to "quiescently solve" any of these problems using a failure detector with finite range, one needs at least $\Diamond\mathcal{P}$.[3]

Let $\mathcal{D}$ be a failure detector with finite range that can be used to solve the Single-Shot Reliable Send and Receive problem with a quiescent algorithm $\mathcal{I}$ ($\mathcal{I}$ is also called the implementation of Send and Receive). We show that $\mathcal{D}$ can be transformed to $\Diamond\mathcal{P}$.

The proof given in this section makes the simplifying assumption that $\mathcal{I}$ does not have an "initialization phase" that requires the sending of messages. In other words, we assume that $\mathcal{I}$ is such that if no process ever Sends any bit, then no process ever sends any messages. This reasonable assumption allows us to simplify the proof and illustrate the basic ideas.

We first give a rough outline of this proof (Section 5.1), and then the proof itself (Sections 5.2 and 5.3). In the Appendix, we give the full proof without the simplifying assumption.

5.1 Intuitive Overview of the Simple Proof

Since $\mathcal{D}$ has a finite range, then for every failure detector history H of $\mathcal{D}$: (a) each failure detector module outputs some values infinitely often (these are the "limit values"), and (b) there is a time after which it outputs only limit values. Let v be a limit value for process p and H. A crucial observation is that with H it is possible to construct runs such that whenever p takes a step it always gets v from its failure detector module. It is easy to generalize the notion of a limit value for p to a limit vector for a set of processes P: A vector f (with a value for every process in the system) is a limit vector for P and H if, for each process p in P, the failure detector module of p outputs $f(p)$ infinitely often in H. Note that with H it is possible to construct runs such that whenever a process p in P takes a step, it obtains $f(p)$ from its failure detector module. We say that vector f hints that P is the set of all correct processes, if f could occur as a limit vector for P when P is the set of correct processes (more precisely, f is a limit vector for P in a history $H \in \mathcal{D}(F)$ where $\text{correct}(F) = P$).

[2] The first letter is capitalized to distinguish them from the lossy send and receive provided by the system.

[3] Moreover, $\Diamond\mathcal{P}$ is sufficient to "quiescently solve" these problems [ACT97] — thus it is the weakest.

Consider a failure detector history H that can occur when P is the set of all correct processes. Let f be any limit vector for P and H. Clearly, f hints that P is the set of all correct processes. Can f also hint that a proper subset P' of P is the set of all correct processes? The answer is no. As we argue next, this is because with $\mathcal{D}$, a process in P' should be able to Send a bit to a process q in $P \setminus P'$ and to do so quiescently using $\mathcal{I}$.

Suppose, for contradiction, that f hints that P' is the set of all correct processes. Then we can construct a run R_1 of $\mathcal{I}$ where (a) P' is indeed the set of all correct processes, (b) processes in P' are scheduled such that whenever they take steps they get f from their failure detector module, (c) some process p in P' Sends a bit b to some process q in $P \setminus P'$, and (d) processes in $P \setminus P'$ never take a step. Because the implementation $\mathcal{I}$ is quiescent, in R_1 eventually all processes in P' (including p) stop sending messages — they give up on trying to transmit b to q.

Since f also hints that P is the set of correct processes, we can create another run R_2 of $\mathcal{I}$ where (a) P is the set of correct processes, (b) processes in P are scheduled such that whenever they take steps they get f from their failure detector module, (c) p Sends b to q, (d) messages sent between processes in P' and processes in $P \setminus P'$ are lost. Note that from the point of view of processes in P', run R_2 is indistinguishable from run R_1. Thus, in R_2 eventually all processes in P' stop sending messages — they give up on trying to transmit b to q. So, in R_2 process q never receives any messages, and thus it does not Receives b from p. Since p and q are correct in R_2, the implementation $\mathcal{I}$ of Send and Receive is incorrect — a contradiction. Thus, f cannot hint that P' is the set of all correct processes.

Let E_P be the set of all vectors that hint that P is the set of correct processes (this set is determined by $\mathcal{D}$). The algorithm that transforms $\mathcal{D}$ to $\Diamond\mathcal{P}$ uses a predetermined "table of hints" containing, for each possible P, the set E_P.

The transformation algorithm works as follows. Each process p periodically sends its current failure detector output to every process, and maintains two variables: f and *Order*. Vector f stores the last failure detector value received from each process, and *Order* is an ordered set of processes. Whenever p receives a failure detector value from another process q, it records that value in $f(q)$ and moves q to the front of *Order*. Let P be the set of correct processes in this run. Note that: (a) eventually f is a limit vector for P, and (b) the correct processes percolate to the front of *Order* (processes that crash end up at the tail), so that eventually P is some prefix of *Order*.

To satisfy the properties of $\Diamond\mathcal{P}$, p must eventually output the complement of P. By (b) above, eventually P is the largest prefix of *Order* that contains correct processes. To find this maximal prefix, p repeatedly uses its current value of f and the predetermined table of hints, as follows. For each prefix P' of *Order*, in order of increasing size, p checks if f hints that P' is the set of all correct processes, i.e., $f \in E_{P'}$, and if so p outputs the complement of P'. This works because, as we argued above, any limit vector f for P: (1) hints that P is the set of all correct processes, and (2) cannot hint that a proper subset P' of P is the set of all correct processes. This concludes the overview of the proof (the reader should understand why the argument above breaks down without the simplifying assumption).

We next give the actual proof. The transformation algorithm $T_{\mathcal{D}\to\Diamond\mathcal{P}}$ uses a table which is determined *a priori* from $\mathcal{D}$ (this is the "table of hints" in our intuitive explanation). We first define this table and show some of its properties (Section 5.2). We then describe and prove the correctness of the transformation algorithm $T_{\mathcal{D}\to\Diamond\mathcal{P}}$ that uses this table (Section 5.3).

5.2 The Predetermined Table

Let $\mathcal{E}$ be an environment and $\mathcal{D}$ be any failure detector with finite range $\mathcal{R} = \{v_1, v_2, \ldots, v_\ell\}$. Let $\mathcal{I}$ be a quiescent implementation of Send and Receive that uses $\mathcal{D}$ in environment $\mathcal{E}$. Assume that if no process Sends any bit then $\mathcal{I}$ does not send any messages (this simplifying assumption is removed in the Appendix).

Given $v_j \in \mathcal{R}$, a process $p \in \Pi$, and a failure detector history H with range $\mathcal{R}$, we say that v_j is *a limit value for p and H* if, for infinitely many t, $H(p,t) = v_j$. Let f be an assignment of failure detector values to every process in Π, i.e., $f : \Pi \longrightarrow \mathcal{R}$. Let P be a non-empty set of processes. We say that f *is a limit vector for P and H* if for all $p \in P$, $f(p)$ is a limit value for p and H. The set of all limit vectors for P and H is denoted $L_P(H)$. Let $E_P^{\mathcal{D},\mathcal{E}} = \{f \mid \exists F \in \mathcal{E}, \exists H \in \mathcal{D}(F) : P = \text{correct}(F) \text{ and } f \in L_P(H)\}$. Roughly speaking, $E_P^{\mathcal{D},\mathcal{E}}$ is the set of limit vectors that could occur when P is the set of correct processes.

The table used by the transformation algorithm $T_{\mathcal{D} \to \Diamond \mathcal{P}}$ consists of all the sets $E_P^{\mathcal{D},\mathcal{E}}$ where P ranges over all non-empty subset of processes. Note that this table is finite. We omit the superscript $\mathcal{D}, \mathcal{E}$ from $E_P^{\mathcal{D},\mathcal{E}}$ whenever it is clear from the context.

Lemma 1. *Let $F \in \mathcal{E}$, $P = correct(F)$ and $H \in \mathcal{D}(F)$. Assume $P \neq \emptyset$. If $f \in L_P(H)$ then $f \in E_P$ and $f \notin E_{P'}$ for every P' such that $\emptyset \subset P' \subset P$.*

Proof. Let $f \in L_P(H)$. The fact that $f \in E_P$ is immediate from the definition of E_P. Let P' be such that $\emptyset \subset P' \subset P$. Suppose, for contradiction, that $f \in E_{P'}$. Then there exists a failure pattern $F' \in \mathcal{E}$ and $H' \in \mathcal{D}(F')$ such that $P' = \text{correct}(F')$ and $f \in L_{P'}(H')$.

We now obtain a contradiction by using the quiescent implementation $\mathcal{I}$ of Send and Receive. Let p be a process in P' and q be a process in $P \setminus P'$. We construct two runs, R_1 and R_2 of $\mathcal{I}$ using $\mathcal{D}$, as follows:

- Run R_1 has failure pattern F' and failure detector history H'. Initially p Sends some bit b to q. Processes in P' take steps and those in $\Pi \setminus P'$ do not. Processes in P' take steps in round-robin fashion such that every time a process $r \in P'$ takes a step, it obtains $f(r)$ from its failure detector module (since $f \in L_{P'}(H')$, $f(r)$ is a limit value for r and H'). Moreover, every process in P' receives every message addressed to it.

  Note that, since $\mathcal{I}$ is quiescent, there is a time t_1 after which no messages are sent or received. Assume without loss of generality that at time t_1 all processes in P' took the same number k of steps (otherwise, choose another time $t_1' > t_1$).

- Run R_2 has failure pattern F and failure detector history H. Initially, processes in R_2 behave as in R_1: p Sends some bit b to q; moreover, each process in P' take the same k steps as in R_1, and process in $\Pi \setminus P'$ do not take any steps. More precisely, processes in P' take steps in round-robin fashion such that every time a process $r \in P'$ takes a step, it obtains $f(r)$ from its failure detector module (since $f \in L_P(H)$ and $r \in P' \subset P$, $f(r)$ is a limit value for r and H). Moreover, every process in P' receives every message addressed to it, and all messages sent to processes in $\Pi \setminus P'$ are lost. This goes on until each process in P' takes k steps, exactly as in R_1.

Let t_2 be the time when this happens. After t_2, processes in P take steps in round-robin fashion such that every time a process $r \in P'$ takes a step, it obtains $f(r)$ from its failure detector module (it does not matter what a process $r \in P \setminus P'$ gets from its failure detector module, as long as it is compatible with H). Moreover, after t_2 no process Sends any bit. This completes the description of run R_2.

Note that at time t_2, each process in P' is in the same state as in run R_1 at time t_1. Moreover, each process in $P \setminus P'$ is in its initial state. By a simple induction argument we can show that after time t_2 in R_2: (a) processes in P' continue to behave as in R_1, (b) processes in $P \setminus P'$ behave as if they were in a run of $\mathcal{I}$ in which no process ever Sends any bit, and (c) no process sends any message (this induction uses the simplifying assumption that in a run in which there are no Sends, no process sends any message). Therefore, in R_2, process q (which is in $P \setminus P'$) never receives any messages. This implies that q does not Receive b from p.

Note that in R_2: (a) both p and q are correct; (b) p Sends b to q; and (c) q does not Receive b from p. Thus, $\mathcal{I}$ is not a correct implementation of Send and Receive — a contradiction. □

5.3 The Transformation Algorithm

The transformation of $\mathcal{D}$ to an eventually perfect failure detector $\mathcal{D}' = \Diamond\mathcal{P}$ in environment $\mathcal{E}$ is shown in Fig. 1. The transformation uses the table of sets E_P (for all non-empty subsets of processes P) that has been determined *a priori* from the given $\mathcal{D}$ and $\mathcal{E}$.

All variables are local to each process. Vector f stores the last failure detector value that p received from each process; *Order* is an ordered set that records the order in which the last failure detector value from each process was received; $\mathcal{D}'_p$ denotes the output of the eventually perfect failure detector that p is simulating (a set of processes that p currently suspects).

In Task 1, each process p periodically sends the output of its failure detector module $\mathcal{D}_p$ to every process q. Upon the receipt of a failure detector value from process q in Task 2, process p enters it into $f[q]$, and moves q to the front of *Order*. Then, p checks if there is some prefix $Order[1..k]$ of *Order* such that $f \in E_{Order[1..k]}$. If there is, it sets $\mathcal{D}'$ to the complement of the smallest such prefix.

We now show that the failure detector constructed by this algorithm, namely $\mathcal{D}'$, is an eventually perfect failure detector. Consider a run of this algorithm with failure pattern $F \in \mathcal{E}$ and failure detector history $H \in \mathcal{D}(F)$, such that $\text{correct}(F) \neq \emptyset$. Let t be the number of processes that crash in F, i.e., $t = |\Pi \setminus \text{correct}(F)|$. Henceforth, p denotes a correct process in F and variables f and *Order* are local to p.

Lemma 2. *There is a time after which (1)* $Order[1..n-t] = correct(F)$*, and (2)* $f \in L_{Order[1..n-t]}(H)$.[4]

Proof. Part (1) is clear from the way *Order* is updated, the fact that p keeps receiving failure detector values from every correct process (recall that only a finite number of messages are lost), and the fact that p eventually stops receiving messages from processes that crash. Part (2) of the lemma follows from part (1) and the fact that the range $\mathcal{R}$ of $\mathcal{D}$ is finite. □

[4] This does not mean that eventually the values of variables f and *Order* at p stop changing. It means that, although they may continue to change forever, eventually the predicates (1) and (2) are true forever at p.

```
1   For every process p:
2
3       Initialization:
4           for all q ∈ Π do f[q] ← ⊥
5           Order ← ∅
6           D'_p ← ∅
7           { For each ∅ ⊂ P ⊆ Π, the set E_P^{D,E} is determined a priori from D and E }
8
9       cobegin
10          || Task 1:
11              repeat periodically
12                  v ← D_p                                          {query D}
13                  for all q ∈ Π do send v to q
14
15          || Task 2:
16              upon receipt of w from q do        {upon receipt of a failure detector value from q}
17                  f[q] ← w
18                  Order ← q || (Order \ {q})          {process q is moved to the front of Order}
19                  if for some k ≥ 1, f ∈ E^{D,E}_{Order[1..k]} then
20                      let k_0 be the smallest such k
21                      D'_p ← Π \ Order[1..k_0]          {suspect processes not in Order[1..k_0]}
22      coend
```

Figure 1: Transformation of $\mathcal{D}$ to an eventually perfect failure detector $\mathcal{D}'$ in environment $\mathcal{E}$

Corollary 3. *There is a time after which (1)* $f \in E_{Order[1..n-t]}$ *and (2) for all* $1 \leq k < n-t$, $f \notin E_{Order[1..k]}$.

Proof. By Lemma 2, there is a time t_0 after which $f \in L_{Order[1..n-t]}(H)$ and $Order[1..n-t] = \text{correct}(F)$. So after time t_0, by Lemma 1, $f \in E_{Order[1..n-t]}$. This shows (1). Let k be such that $1 \leq k < n-t$. After t_0, $\emptyset \subset Order[1..k] \subset \text{correct}(F)$, and $f \in L_{\text{correct}(F)}(H)$. So, by Lemma 1, $f \notin E_{Order[1..k]}$. This shows (2). □

Corollary 4. *There is a time after which* $\mathcal{D}'_p = \Pi \setminus correct(F)$.

Proof. By Corollary 3 and the algorithm, there is a time after which the k_0 selected in line 20 is always $n-t$. Now apply Lemma 2 part (1). □

By Corollary 4, we have:

Theorem 5. *Consider an asynchronous distributed system with process crashes and links that may lose a finite number of messages. Suppose failure detector* $\mathcal{D}$ *with finite range can be used to solve the Single-Shot Reliable Send and Receive problem in environment* $\mathcal{E}$*, and that the implementation is quiescent. Assume further that if no process ever Sends any bit then this implementation does not send any messages. Then* $\mathcal{D}$ *can be transformed (in environment* $\mathcal{E}$*) to the eventually perfect failure detector* $\Diamond\mathcal{P}$.

Acknowledgments

We are very grateful to Tushar Deepak Chandra for his extensive comments. In particular, his insightful comments regarding the simplifying assumption of Theorem 5 helped us write the Appendix.

References

[ACT97] Marcos Kawazoe Aguilera, Wei Chen, and Sam Toueg. Heartbeat: a timeout-free failure detector for quiescent reliable communication. Technical Report 97-1631, Department of Computer Science, Cornell University, May 1997.

[BCBT96] Anindya Basu, Bernadette Charron-Bost, and Sam Toueg. Simulating reliable links with unreliable links in the presence of process crashes. In *Proceedings of the 10th International Workshop on Distributed Algorithms*, Lecture Notes on Computer Science, pages 105–122. Springer-Verlag, October 1996.

[CHT96] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, July 1996.

[CT96] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

A Appendix

We now give an extended, more complex proof of our main result without the simplifying assumption of Theorem 5.

Let $\mathcal{E}$ be an environment and $\mathcal{D}$ be any failure detector with finite range $\mathcal{R} = \{v_1, v_2, \ldots, v_\ell\}$. Let $\mathcal{I}$ be a quiescent implementation of Send and Receive that uses $\mathcal{D}$ in environment $\mathcal{E}$.

As in the simpler proof in Section 5, the transformation algorithm $T_{\mathcal{D} \to \Diamond \mathcal{P}}$ uses a finite table that is predetermined from $\mathcal{D}$. We first define this table and show some of its properties (Section A.1). We then describe and prove the correctness of the transformation algorithm $T_{\mathcal{D} \to \Diamond \mathcal{P}}$ that uses this table (Section A.2).

A.1 The Predetermined Table

For the definitions in this proof, let:

- v_j be a failure detector value, i.e., $v_j \in \mathcal{R}$
- p be a process, i.e., $p \in \Pi$
- F be a failure pattern
- H be a failure detector history with range $\mathcal{R}$
- f be an assignment of failure detector values to every process in Π, i.e., $f : \Pi \longrightarrow \mathcal{R}$
- P and P_0 be non-empty set of processes
- $p_0, p_1, \ldots, p_{m-1}$ be the processes in P (where $m = |P|$ and $p_0 < p_1 < \cdots < p_{m-1}$)

Definition 1. We say that v_j is *a limit value for p and H* if, for infinitely many t, $H(p, t) = v_j$.

Definition 2. We say that *f is a limit vector for P and H* if for all $p \in P$, $f(p)$ is a limit value for p and H. The set of all limit vectors for P and H is denoted $L_P(H)$.

Definition 3. *RRIRounds*(P, f) is defined as follows.

Consider the round-robin execution of implementation $\mathcal{I}$ in which: (a) processes in P take steps forever in a round-robin fashion[5] and processes in $\Pi \setminus P$ do not take any steps, (b) no process ever Sends any bit, (c) every time a process $p \in P$ queries its failure detector module, p gets $f(p)$, (d) every time a process $p \in P$ takes a step, p receives the earliest message sent to it that it did not yet receive (thus, every $p \in P$ eventually receives each message addressed to it), and (e) all messages sent to processes in $\Pi \setminus P$ are lost.[6]

There are two possible cases in the above round-robin execution of $\mathcal{I}$:

- *Every process eventually stops sending messages*. In this case, after some number k of round-robin rounds, no process ever receives any messages. We say that "round-robin initialization (r.r.i.) occurs in k rounds", and define *RRIRounds*$(P, f) = k$.
- *Some process never stops sending messages*. In this case, we define *RRIRounds*$(P, f) = \infty$.

[5] That is, p_0 takes the first step, then p_1 takes a step, and so on, so that the j-th step is taken by process $p_{(j-1) \bmod m}$.

[6] It is possible that this is *not* a valid execution of $\mathcal{I}$ using $\mathcal{D}$ in environment $\mathcal{E}$.

Intuitively, we say that F and H allow round-robin initialization for P and f if the following holds: (a) in the above execution with P and f, r.r.i. occurs in k rounds for some k, and (b) there is a schedule compatible with F and H that allows this k-round r.r.i. More precisely:

Definition 4. We say that *F and H allow round-robin initialization (r.r.i.) for P and f* if (a) *RRIRounds*$(P, f) = k$ for some k, and (b) if there are times $t_0 < t_1 < \cdots < t_{mk-1}$ such that for every $0 \leq j \leq mk-1$, (1) $p_{j \bmod m}$ is not crashed at time t_j, i.e., $p_{j \bmod m} \notin F(t_j)$ and (2) the failure detector module of $p_{j \bmod m}$ at time t_j outputs $f(p_{j \bmod m})$, i.e., $H(p_{j \bmod m}, t_j) = f(p_{j \bmod m})$.

Definition 5. $L_{P,P_0}(F, H) = \{f \mid f \in L_P(H)$ and F and H allow r.r.i. for P_0 and $f\}$.

Definition 6. $E^{\mathcal{D},\mathcal{E}}_{P,P_0} = \{f \mid \exists F \in \mathcal{E}, \exists H \in \mathcal{D}(F) : P = \text{correct}(F)$ and $f \in L_{P,P_0}(F, H)\}$.

Roughly speaking, $E^{\mathcal{D},\mathcal{E}}_{P,P_0}$ is the set of limit vectors f that could occur when P is the set of correct processes and it is possible to have r.r.i. for P_0 and f.

The table used by the transformation algorithm $T_{\mathcal{D} \to \Diamond \mathcal{P}}$ consists of all the sets $E^{\mathcal{D},\mathcal{E}}_{P,P_0}$ where P and P_0 range over all non-empty subset of processes. Note that this table is finite. We omit the superscript $\mathcal{D}, \mathcal{E}$ from $E^{\mathcal{D},\mathcal{E}}_{P,P_0}$ whenever it is clear from the context.

Lemma 6. *Let $F \in \mathcal{E}$, $P = correct(F)$, $H \in \mathcal{D}(F)$ and $f \in L_P(H)$. Assume $P \neq \emptyset$. Then RRIRounds$(P, f) < \infty$.*

Proof. We can construct a run R of implementation $\mathcal{I}$ using $\mathcal{D}$ with $F \in \mathcal{E}$, such that all processes behave exactly as in the round-robin execution of $\mathcal{I}$ that was used to define *RRIRounds*(P, f). To see this, note that since $F \in \mathcal{E}$, $P = \text{correct}(F)$, $H \in \mathcal{D}(F)$ and $f \in L_P(H)$, we can find times for the round-robin steps of correct processes such that, for each time u at which a process p takes a step, the output $H(p, u)$ of its failure detector module is $f(p)$. Since $\mathcal{I}$ is quiescent, there is a time after which no process sends any message in run R. Thus, *RRIRounds*$(P, f) < \infty$. □

Lemma 7. *Let $F \in \mathcal{E}$, $P = correct(F)$, $H \in \mathcal{D}(F)$. Assume $P \neq \emptyset$ and let P_0 be such that $P \subseteq P_0 \subseteq \Pi$. If $f \in L_{P,P_0}(F, H)$ then $f \in E_{P,P_0}$ and $f \notin E_{P',P_0}$ for all P' such that $\emptyset \subset P' \subset P$.*

Proof. Let $f \in L_{P,P_0}(F, H)$. The fact that $f \in E_{P,P_0}$ is immediate from the definition of E_{P,P_0}. Let P' be such that $\emptyset \subset P' \subset P$. Suppose, for contradiction, that $f \in E_{P',P_0}$. Then there exists a failure pattern $F' \in \mathcal{E}$ and $H' \in \mathcal{D}(F')$ such that $P' = \text{correct}(F')$ and $f \in L_{P',P_0}(F', H')$.

We now obtain a contradiction by using the quiescent implementation $\mathcal{I}$. Let p be a process in P' and q be a process in $P \setminus P'$. We construct three runs of $\mathcal{I}$, namely, R_0, R_1 and R_2. Roughly speaking, each one of these runs starts with an r.r.i. for P_0 and f. After this initialization, in R_0 nothing else happens, in R_1 process p Sends some bit to q but q crashes, and in R_2 process p Sends the same bit to q and q is correct. We will reach a contradiction by arguing that in R_2 process q behaves as in R_0, and thus it never Receives any bit from p — this violates the defining property of Send and Receive.

Runs R_0, R_1 and R_2 are defined as follows:

- Run R_0 has failure pattern F and failure detector history H. Since $f \in L_{P,P_0}(F,H)$, $f \in L_P(H)$, and F and H allow r.r.i. for P_0 and f. R_0 consists initially of a r.r.i. for P_0 and f. More precisely, initially: (a) processes in P_0 take steps in a round-robin fashion and processes in $\Pi \setminus P_0$ do not take any steps, (b) no process Sends any bit, (c) every time a process $r \in P_0$ queries its failure detector module, r gets $f(r)$, (d) every time a process $r \in P_0$ takes a step, r receives the earliest message sent to it that it did not yet receive, and (e) all messages sent to processes in $\Pi \setminus P_0$ are lost. This goes on until each process in P_0 has taken $\mathit{RRIRounds}(P_0, f)$ steps. Let t_0 be the time when this happens. After t_0, processes in P take steps in a round-robin fashion such that every time a process $r \in P$ takes a step, it obtains $f(r)$ from its failure detector module (this is possible because $f \in L_P(H)$); moreover, no process Sends any bit.

  Note that since both p and q are in $P = \text{correct}(F)$, and p does not Send any bit to q, it must be that q does not Receive any bit from p. Furthermore, after time t_0, no processes send or receive any messages.

- Run R_1 has failure pattern F' and failure detector history H'. Since $f \in L_{P',P_0}(F',H')$, $f \in L_{P'}(H')$, and F' and H' allow r.r.i. for P_0 and f. Initially, processes in R_1 behave as in R_0, i.e., R_1 starts with a r.r.i. for P_0 and f. Then, execution proceeds as follows: (a) p Sends some bit b to q, (b) processes in P' take steps in round-robin fashion and processes in $\Pi \setminus P'$ take no steps, (c) every time a process $r \in P'$ takes a step, it obtains $f(r)$ from its failure detector module, (d) every time a process $r \in P'$ takes a step, r receives the earliest message sent to it that it did not yet receive, (e) all messages sent to processes in $\Pi \setminus P'$ are lost.

  Note that, since implementation $\mathcal{I}$ is quiescent, there is a time t_1 after which no messages are sent or received. Assume without loss of generality that at time t_1 every process in P' took the same number k of steps (otherwise, choose another time $t_1' > t_1$).

- Run R_2 has failure pattern F and failure detector history H. Initially, processes in R_2 behave as in R_1: R_2 starts with a r.r.i. for P_0 and f, and then p Sends b to q and execution continues as in R_1, until each process in P' has taken k steps (this is possible because $f \in L_P(H)$ and $P' \subseteq P$).

  Let t_2 be the time when this happens. After t_2, execution proceeds as follows: (a) no process Sends any bit, (b) processes in P take steps in round-robin fashion and processes in $\Pi \setminus P$ take no steps, (c) every time a process $r \in P$ takes a step, it obtains $f(r)$ from its failure detector module (this is possible because $f \in L_P(H)$).

  Note that at time t_2, each process in P' is in the same state as in run R_1 at time t_1, and each process in $P \setminus P'$ is in the same state as in run R_0 at time t_0. A simple induction on the steps taken shows that, in R_2, (1) processes in P' have the same behavior as in run R_1; (2) processes in $P \setminus P'$ have the same behavior as in run R_0; (3) no messages are sent or received after time t_2. Since $q \in P \setminus P'$ and q does not Receive any bit from p in R_0, it does not Receive any bit from p in R_2.

In summary, in R_2: (a) both p and q are correct; (b) p Sends b to q; and (c) q does not Receive b from p. Thus, $\mathcal{I}$ is not a correct implementation of Send and Receive — a contradiction. □

A.2 The Transformation Algorithm

The transformation of $\mathcal{D}$ to an eventually perfect failure detector $\mathcal{D}' = \Diamond\mathcal{P}$ in environment $\mathcal{E}$ is shown in Fig. 2. The transformation uses the table of sets E_{P,P_0} (for all non-empty subsets P and P_0 of processes) that has been determined *a priori* from the given $\mathcal{D}$ and $\mathcal{E}$.

All variables are local to each process. *Sequences* is a finite set of finite sequences of pairs (p, v) where $p \in \Pi$ is a process and $v \in \mathcal{R}$ is a failure detector value. It stores possible schedules that could have resulted from F and H. Vector f stores the last failure detector value that p received from each process. *Order* is an ordered set that records the order in which the last failure detector value from each process was received. $\mathcal{D}'_p$ denotes the output of the eventually perfect failure detector that p is simulating (a set of processes that p currently suspects). *AllowsRRI* is a boolean function that takes three parameters: a set *Sequences*, a set $P = \{p_0, p_1, \ldots, p_{m-1}\} \subseteq \Pi$ (where $p_0 < p_1 < \cdots < p_{m-1}$), and a vector f. It returns true if and only if for some sequence $s \in$ *Sequences*, there exists a subsequence of s that consists of *RRIRounds*(P, f) repetitions of $(p_0, f(p_0)), (p_1, f(p_1)), \ldots, (p_{m-1}, f(p_{m-1}))$.

In Task 1, each process p periodically queries its failure detector module, appends a new pair to each sequence in *Sequences* and then sends *Sequences* and the output of its failure detector module $\mathcal{D}_p$ to every process. Upon the receipt of $(Sequences', v')$ from process q in Task 2, process p enters v' into $f[q]$, moves q to the front of *Order*, and updates *Sequences*. Then, p uses the function *AllowsRRI* to check whether there is some k such that r.r.i. could have occurred for $Order[1..k]$ and f. If there is, it sets k_0 to the *largest* such k, and then checks if for some k', $f \in E_{Order[1..k'],Order[1..k_0]}$. If so, it sets k_1 to the *smallest* such k', and sets $\mathcal{D}'$ to the complement of $Order[1..k_1]$.

We now show that the failure detector constructed by this algorithm, namely $\mathcal{D}'$, is an eventually perfect failure detector. Consider a run of this algorithm with failure pattern $F \in \mathcal{E}$ and failure detector history $H \in \mathcal{D}(F)$, such that correct$(F) \neq \emptyset$. Let t be the number of processes that crash in F, i.e., $t = |\Pi \setminus \text{correct}(F)|$. Henceforth, p denotes a correct process in F, and f, *Order*, and *Sequences* are variables local to p.

Lemma 8. *There is a time t_0 after which (1) $Order[1..n-t] = correct(F)$, (2) $f \in L_{Order[1..n-t]}(H)$ and (3) AllowsRRI$(Sequences, Order[1..n-t], f)$.*[7]

Proof. Note that p eventually stops receiving messages from processes that crash, and p never stops receiving messages from correct processes (recall that only a finite number of messages are lost). From the way *Order* is updated, there is a time t_1 after which (1) holds.

Let $P = \text{correct}(F)$. Variable f ranges over a finite number of values, so there are functions $f_1, f_2, \ldots, f_N : \Pi \to \mathcal{R}$ such that (a) for every $1 \leq j \leq N$, variable f is equal to f_j an infinite number of times, and (b) there is a time t_2 after which the predicate $f \in \{f_1, f_2, \ldots, f_N\}$ holds. We now show that for every $1 \leq j \leq N$, $f_j \in L_P(H)$, and there is a time τ_j after which *AllowsRRI*$(Sequences, P, f_j)$ holds. Together with (1) and (b), this implies that after time $t_0 = \max\{t_1, t_2, \tau_1, \tau_2, \ldots, \tau_N\}$, both (2) and (3) hold.

Let $1 \leq j \leq N$. We first claim that each process $q \in P$ obtains $f_j(q)$ from $\mathcal{D}$ in line 13 an

[7] This does not mean that eventually the values of variables f, *Sequences*, and *Order* at p stop changing. It means that, although they may continue to change forever, eventually the predicates (1), (2) and (3) are true forever at p.

```
1   For every process p:
2
3       Initialization:
4           for all q ∈ Π do f[q] ← ⊥
5           Order ← ∅
6           Sequences ← {λ}
7           D'_p ← ∅
8           { For each ∅ ⊂ P, P_0 ⊆ Π, the set E^{D,E}_{P,P_0} is determined a priori from D and E }
9
10      cobegin
11          || Task 1:
12              repeat periodically
13                  v ← D_p                                              {query D}
14                  append (p, v) to each sequence in Sequences
15                  for all q ∈ Π do send (Sequences, v) to q
16
17          || Task 2:
18              upon receipt of (Sequences', v') from q do
19                  f[q] ← v'
20                  Order ← q || (Order \ {q})          {process q is moved to the front of Order}
21                  Sequences ← Sequences ∪ Sequences'
22                  if for some k ≥ 1, AllowsRRI(Sequences, Order[1..k], f) then
23                      let k_0 be the largest such k
24                      if for some k' ≥ 1, f ∈ E^{D,E}_{Order[1..k'],Order[1..k_0]} then
25                          let k_1 be the smallest such k'
26                          D'_p ← Π \ Order[1..k_1]          {suspect processes not in Order[1..k_1]}
27      coend
```

Figure 2: Transformation of $\mathcal{D}$ to an eventually perfect failure detector $\mathcal{D}'$

infinite number of times — this immediately implies $f_j \in L_P(H)$. To show the claim, since only a finite number of messages are lost, process p receives a message from q and updates $f[q]$ an infinite number of times. Together with (a), this implies that p receives a message containing $f_j(q)$ from q an infinite number of times, and this implies the claim.

We now show that there is a time τ_j after which *AllowsRRI*$(Sequences, P, f_j)$ holds. Let T be the time after which no message sent to a correct process is lost. Since $f_j \in L_P(H)$, by Lemma 6, *RRIRounds*$(P, f_j) = k$ for some $k < \infty$. Let $p_0 < p_1 < \cdots < p_{m-1}$ be the processes in P. By the claim, at some time $u_0 > T$, p_0 obtains $f_j(p_0)$ from $\mathcal{D}$ in line 13. After doing so, p_0 appends $(p_0, f_j(p_0))$ to all sequences in *Sequences* and sends a message containing *Sequences* to all processes. At some time $u_1' > u_0$, p_1 receives this message and updates *Sequences*. By the claim, at some time $u_1 > u_1'$, p_1 obtains $f_j(p_1)$ from $\mathcal{D}$ in line 13. After doing so, p_1 appends $(p_1, f_j(p_1))$ to all sequences in *Sequences* and so p_1 obtains a sequence containing $(p_0, f_j(p_0))$ before $(p_1, f_j(p_1))$. We can repeat this argument for all the processes in P in a round-robin order, for $k+1$ rounds, and conclude that eventually *AllowsRRI*$(Sequences, P, f_j)$ holds. □

Lemma 9. *There is a time t_1 after which for every $m_0 \geq n - t$ such that AllowsRRI(Sequences, Order$[1..m_0], f$) holds: (1) $f \in E_{Order[1..n-t],Order[1..m_0]}$ and (2) for all $1 \leq m_1 < n - t$, $f \notin E_{Order[1..m_1],Order[1..m_0]}$.*

Proof. By Lemma 8, there is a time t_0 after which (a) $Order[1..n-t] = \text{correct}(F)$, and (b) $f \in L_{Order[1..n-t]}(H)$. Let $t_1 = t_0$. Suppose that at some time $t'_1 > t_1$, *AllowsRRI*(*Sequences*, *Order*$[1..m_0], f$) holds for some $m_0 \geq n-t$. This implies that F and H allow r.r.i. for *Order*$[1..m_0]$ and f. From (b), $f \in L_{Order[1..n-t],Order[1..m_0]}(F, H)$ holds at time t'_1. Thus, by Lemma 7, $f \in E_{Order[1..n-t],Order[1..m_0]}$.

Let $1 \leq m_1 < n-t$. By (a), $\emptyset \subset Order[1..m_1] \subset \text{correct}(F) \subseteq Order[1..m_0]$ holds at time t'_1. Note that $f \in L_{Order[1..n-t],Order[1..m_0]}(F, H)$ holds at time t'_1. By Lemma 7, $f \notin E_{Order[1..m_1],Order[1..m_0]}$. □

Corollary 10. *There is a time after which $\mathcal{D}'_p = \Pi \setminus correct(F)$.*

Proof. By Lemma 8 part (3), there is a time t_0 after which every time p receives some message, the **if** in line 22 evaluates to true and the k_0 selected in line 23 is at least $n - t$. After time t_0, by Lemma 9, there is a time after which: every time p receives some message, the **if** in line 24 evaluates to true and the k_1 selected in line 25 is $n - t$. Now apply Lemma 8 part (1). □

By Corollary 10, we have:

Theorem 11. *Consider an asynchronous distributed system with process crashes and links that may lose a finite number of messages. Suppose failure detector $\mathcal{D}$ with finite range can be used to solve the Single-Shot Reliable Send and Receive problem in environment $\mathcal{E}$, and that the implementation is quiescent. Then $\mathcal{D}$ can be transformed (in environment $\mathcal{E}$) to the eventually perfect failure detector $\Diamond\mathcal{P}$.*